Component Integrity Guarantees in Software-Defined Networking Infrastructure

Daniel Girtler
dagi5259@student.su.se
Department of
Computer and Systems Sciences
Stockholm Universiy

Nicolae Paladi
nicolae.paladi@ri.se
Security Lab
SICS RISE

***Abstract*—Operating system level virtualization containers are commonly used to deploy virtual network functions (VNFs) which access the centralized network controller in software-defined networking (SDN) infrastructure. While this allows flexible network configuration, it also increases the attack surface, as sensitive information is transmitted between the controller and the virtual network functions. In this work we propose a mechanism for bootstrapping secure communication between the SDN controller and deployed network applications. The proposed mechanism relies on platform integrity evaluation and execution isolation mechanisms, such as Linux Integrity Measurement Architecture and Intel Software Guard Extensions. To validate the feasibility of the proposed approach, we have implemented a proof of concept which was further tested and evaluated to assess its performance. The prototype can be seen as the first step into providing users with security guarantees regarding the integrity of components in the SDN infrastructure.**

***Keywords*—SDN, NFV, SGX, IMA, Docker, security**

I. Introduction

A growing reliance on server virtualization and cloud computing mandate a revision of traditional network architecture models [1]. Enterprise data centers and carrier infrastructures require flexible and scalable network resource provisioning [2]. Software-defined networking (SDN) aims to address these needs by decoupling the data plane from the control plane, thus providing a more dynamic, manageable and adaptable architecture [3]. However, the SDN approach introduces novel vulnerabilities unaddressed by earlier best practices [4], [5]. While several approaches have been proposed to both detect attacks on SDN infrastructure [6] and secure the communication between the control plane and the virtual network functions (VNFs) [7], verification of the integrity of VNFs has not been addressed in much detail so far [8]. Thus, in the event of a compromise of a VNF, the adversary can communicate directly to the controller, impersonate legitimate VNFs and exploit vulnerabilities in its application programming interface. Furthermore, in the context where an SDN controller is accessed by VNFs executing on multiple remote platforms, it is essential to prevent the enrollment of malicious VNFs and only allow communication once the VNF has been explicitly enrolled.

In this work, we address the current lack of mechanisms to establish the trustworthiness of VNFs and protect their authentication credentials. We describe a mechanism to bootstrap trust in VNFs deployed on remote platforms and maintain control over the sensitive data, such as VNF authentication credentials, in the face of a powerful adversary. Thus, the SDN controller may monitor the trustworthiness of the VNF and provision or revoke VNF authentication credentials, depending on the monitoring results. The proposed mechanism builds on recent advances in hardware-assisted isolated execution environments and prevents attacks on the integrity of the VNFs, as well as integrity and confidentiality of communication between the VNFs and network controllers.

a) Contribution: Our contribution is as follows: we describe, implement and evaluate a comprehensive mechanism allowing to *attest the integrity of the remote system*, *measure and monitor the integrity of the application* and *protect the authentication credentials* provisioned to enrolled VNFs.

b) Structure: The remaining of the paper is structured as follows: following a review of the system model in §II, we define the adversary model §III, describe the system design in §IV as well as detail a fully functional prototype of the designed system §V. Finally, we evaluate the proposed mechanism in §VI, review the related work in §VII, point out future work directions and conclude in §VIII.

II. Background

We briefly review the system model and introduce the building blocks used in the proposed security mechanism.

a) Software-Defined Networking: The SDN architecture model decouples the data plane from the control plane, to improve flexibility and manageability. According [3], SDN is defined by the following characteristics: (1) separation of control and data planes; (2) flow-based (rather than destination based) forwarding decisions, based on matches in a predefined filter, which allows for a unification of network devices, such as firewalls, switches, and routers; (3) the control logic defining the flow mechanisms operates on a distinct entity, a so-called *SDN controller*; (4) virtual network functions deployed of the controller can program the network;

In this work we focused on the application plane, consisting of VNFs accessing the control plane. We aim to protect the security-sensitive data transferred during such access, in order to create a trust relationship between the SDN controller and a VNF enrolled in the SDN deployment.

b) Network Function Virtualization (NFV): The type of VNFs deployed in the application plane is deployment-specific and may include firewalls, proxies, or DHCP servers. Rather

978-1-5386-3285-7/17/$31.00 ©2017 IEEE

than deploying custom-built hardware components, network function virtualization implements network services in software, such that they can be run on commodity platforms [9]. NFV enables flexible and dynamic infrastructures, allowing applications to be launched and torn down as needed. In some implementations, VNFs are integrated within the controller. A different approach – also chosen as the system model in this work – is to deploy them on a discreet platform communicating remotely with the controller.

c) Operating-system virtualization: Operating system virtualization allows to concurrently execute multiple isolated execution environments, referred to as containers [10]. Containers rely on the functionality of a common kernel, which ensure the isolation between containers[1]. Containers can be created and destroyed on demand, allowing thus flexibility and resource efficiency. VNFs can be deployed inside such isolated containers, leading to a flexible way of deploying and operating network services.

d) Linux IMA: The first step in assessing the trustworthiness of a remote platform is to verify its integrity. To accomplish this, we enable platform integrity measurements using Linux IMA[2], which allows to detect if files have been accidentally or maliciously altered, by comparing a file's measurement against an expected value. For each file to be measured, a SHA-256 hash is calculated and stored in a list. To ensure that this measurement list is not modified at any point, it can be stored in a dedicated security hardware module, e.g. a Trusted Platform Module [11]. Trustworthiness is assessed following a *remote attestation* protcol, where the integrity measurements are transferred to the attester which assesses platform integrity by comparing them to pre-calculated *expected* measurements.

e) Software Guard Extenstion (SGX): Intel SGX [12] is a technology for creating hardware-assisted isolated execution environments (termed *enclaves*). Its functionality allows to protect both the integrity and confidentiality of code and data in the enclaves. SGX is implemented based on an extension of the Instruction Set architecture and changes to memory access, allowing to create enclaves located inside an application's address space, inaccessible to the underlying operating system and BIOS. Code executed in enclaves runs in user space and relies on the underlying operating system for I/O operations and system calls. SGX also includes functionality for remote attestation, allowing to remotely assess the integrity of an enclave. However, remote attestation of enclaves relies on communication with an additional component – the *Intel Attestation Service* (IAS) – to verify that enclave integrity measurements are signed with valid (i.e. not revoked) keys.

III. Adversary Model

We defined an adversary model based on the Dolev-Yao model [13], which states that the adversary can *intercept*, *eavesdrop* and *tamper* with messages. In addition to these abilities, the adversary can disrupt or degrade network traffic [14]. Further adversary abilities are grouped as follows:

a) Networking: The adversary can record, intercept and replay messages sent over the network. This is restricted to communication between the controller host and the VNF host and does not include network communication between applications residing on the controller platform. This ability is further constrained by any applied cryptographic functions.

b) Remote system: The adversary is able to gain access to the remote VNF host by exploiting vulnerabilities in the operating system and has control of all software running on the host. On the hardware level, we assume the CPU is implemented correctly and that the physical security of the platform is maintained. All remaining hardware, including the memory, can be tempered by the adversary.

IV. System Design

We next review the core principles of the system design.

a) Phase one: Measurement scheme: In the first step of the attestation process we verify that the remote host has not been compromised. To establish a trust relationship between the SDN controller and the remote VNF, we design a specific Linux IMA policy to measure a restricted set of files critical for the secure execution of the VNFs on the remote host (i.e. its *trusted computing base*).

b) Phase two: Integrity assessment: Evaluating the measurements of the trusted computing base allows to detect VNF compromise and halt the protocol execution. In this phase, the measurements collected in phase one are transferred to the controller and compared to an expected value, to thereby determine whether the trusted computing base of the remote host has been been compromised. Execution proceeds only if the measurements match the expected values.

c) Phase three: Verifying VNFs: Following a successful completion of phase two, the controller would establish a communication channel with the VNF deployed on the remote system. A complication arises if the VNF is running on a host under the control of the adversary and that has previously not been verified (a so-called *cuckoo-attack* [15]). We address this by ensuring that the VNF executes on the verified remote host. The VNF may access the SDN controller only after successfully verifying that the target's respective VNF resides on an attested platform.

d) Phase four: Content trust of images: Container images can be either created on a local system or obtained from remote image repositories. In the first case, the owner is solely responsible for their correct configuration. However, this does not apply to images from remote repositories. Instead, the content of such images must be integrity verified against measurements of valid image configurations.

V. Implementation

We next describe the implementation of the proposed design. For the prototype implementation we used Ubuntu 16.04LTS with kernel version 4.4.0-51-generic for both controller and the VNF host. For operating system virtualization management, we used Docker v1.12.2, a popular container management software [16]. An open-source SDN controller, *Floodlight* version 1.2, was chosen due to its support for different security modes for the REST API, non-secure (plain vanilla

[1]Docker project: https://docs.docker.com
[2]https://sourceforge.net/p/linux-ima/wiki/Home

HTTP), HTTPS and trust-based HTTPS [3]. For serialization of network-communicated data we use language and platform neutral protocol buffers[4].

A. Implementation of the system design

To limit the measurements collected with Linux IMA, we measure only core Docker assets, focusing on Docker related executables in the */usr/bin/* directory and security policy related files used by AppArmor in */etc/apparmor.d/*. Furthermore, images and metadata used by Docker were incldued into the integrity measurements to guarantee their integrity – both as a whole and of single layers of images.

The next step requires transferring the measurement list from the VNF host to the SDN controller for validation. This is achieved by leveraging the SGX remote attestation mechanism, namely by using the Diffie-Hellman Key Exchange (DHKE) protocol provided by SGX to create secure keys for the data encryption. A software component is deployed in an SGX enclave executing on the VNF host, which retrieves the measurement file and sends it to the controller performing the validation process. To reduce the prototype complexity, we stored the measurement list on the file system. We used *mbedtls-SGX*[5] TLS protocol suite to establish a secure channel for the transfer of measurements to the controller.

Validating that the VNF is running on the attested host can be done by creating a non-migratable HMAC key K [17], stored in a hardware security model on the VNF host (e.g. a Trusted Platform Module, TPM). We send K and a nonce $n1$ to the remote system's enclave. The enclave software calculates a hash-based message authentication code (HMAC) of $n1$ and K and returns the resulting HMAC to the controller. The controller will first verifies the HMAC received from the VNF host and if valid, generate a nonce $n2$, and send it to the target VNF. The VNF is able to access the previously stored key K in the TPM, if it resides on the same platform. Next the HMAC of $n2$ and K is computed inside the VNF enclave and returned to the controller. Since only a co-residing VNF can have access to the TPM were K was stored and calculate the correct HMAC, it can be determined that the VNF is indeed running on the same host platform. Next, authentication certificates are generated and signed for successfully attested VNFs, to enable client authentication. The validity of the signed certificates is verified by the SDN controller prior establishing a TLS communication channel. Attempts for insecure communication sessions are refused by the SDN controller.

We leverage Docker functionality for image verification and integrity checking, known as Docker Content Trust (DCT). DCT ensures that the VNF host only runs trusted images – i.e. that the host is only able to pull, run or build signed new images. Prior to establishing a communication channel between the VNFs and the network controller, the VNF host must prove that DCT is enabled. This is done by including the DCT environment variable into the measurement list, verified in the earlier stages.

[3]Trust-based HTTPS means that client authentication is required.

[4]Google Protocol Buggers https://developers.google.com/protocol-buffers/

[5]https://github.com/bl4ck5un/mbedtls-SGX

B. System workflow

We introduce a *Verification Manager* to perform the attestation tasks. It handles the necessary communication with the IAS, generates the HMAC key and nonces, as well as certificates for client authentication. The controller is connected via a TLS channel to the VNF host and the VNFs deployed in containers. We next describe the system workflow (Figure 1).

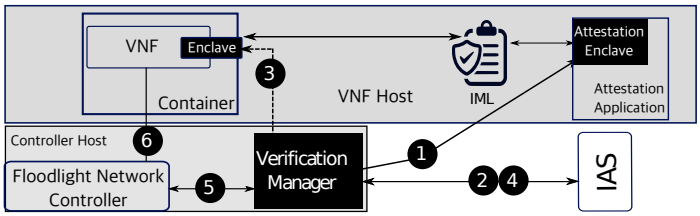

Fig. 1. Workflow of the developed system

The workflow starts at the verification manager (1) which initiates the remote attestation of the VNF host; this involves the DHKE message exchange during which the IAS is contacted (2). Following a successful remote attestation, the measurement list is sent for verification from the VNF host to the verification manager, which then generates a key and a nonce sent back to the VNF host along with the attestation result. In turn, the VNF host calculates the HMAC and sends it back to the verification manager. After the remote system attestation has been completed successfully, the manager attests the integrity of the VNF enclave (3), which requires a second contact with the IAS (4). Following the completion of the DHKE, the verification manager sends a second nonce to the VNF, which returns the computed HMAC. After the verification of this HMAC, the manager generates the certificate and private key and distributes them to the controller and the VNF (5). Once the VNF receives the certificate and private key, it can start the TLS communication with the controller (6). Note that the current implementation of the prototype does not use a hardware root of trust (e.g. a TPM) yet. Support for TPM is currently in progress and when complete will be used to store the measurement list and the HMAC key.

VI. Performance Evaluation

We evaluated the implemented prototype on a Thinkpad T460s with SGX support as VNF host and a Thinkpad E540 running Floodlight and the verification manager. We used *curl* for HTTP connections between the two machines; HTTPS and trusted HTTPS connections were established by using the *mbedtls* library.

The connection from within the SGX enclave was established by using a version of mbedtls with SGX support, with a measurement based on 50 samples. As shown in Figure 2, the HTTP request has the lowest execution time (49 ms on average). The average of the HTTPS and the trusted HTTPS is about 77 ms higher than HTTP, completing at 127 ms. The mbedtls version with SGX support was faster than HTTPS and trusted HTTPS, completing on average in 101 ms.

Figure 3 illustrates the execution time of the attestation process, which completes in 3512 ms. Most of this time is spent on

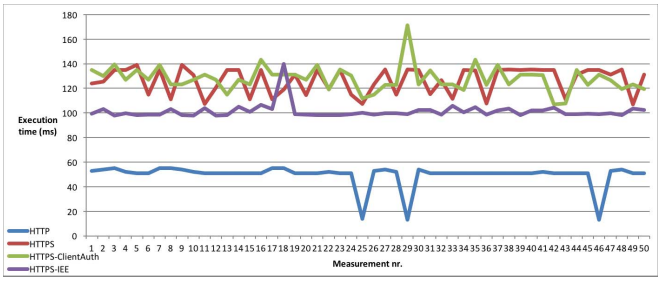

Fig. 2. Executions times of connecting to the Floodlight REST API

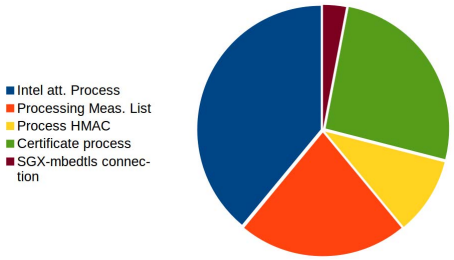

Fig. 3. Parts of overall execution time

the attestation of SGX enclaves (1390 ms), generation/storage of the certificates for the client authentication (905 ms), transfer/validation of the measurement list (763 ms) and the generation/validation process of the HMAC key (353 ms). In contrast with the above time-consuming tasks, the connection to Floodlight requires only about 101 ms.

VII. Related Work

Initial work for bootstrapping trust in SDN infrastructures using SGX was done in [18], which presents a framework for isolating network endpoints in SGX enclaves attested and verified before establishing secure communication channels. Shih et al [19] proposed to protect sensitive components of intrusion detection systems using Intel SGX. However, in this case SGX had been used for very specific NFVs with a limited application field. SGX is a novel technology under continuous development and intense security research scrutiny.

[20] exposes vulnerability exploits synchronization bugs to circumvent the security guarantees of SGX and to hijack the control flow of the enclave code or even bypass access control mechanisms. In [21], Buhren et al described a so-called *fault attack*, allowing an attacker to change the contents of the RAM encrypted area. The adversary might not know which of the encrypted values have been changed or what the impact of it is, but the paper addresses the question, if encryption is a dependable protection mechanism in practice. In addition to this, a proof-of-concept was developed which shows the process of the fault attack and eventually led to the extraction of the private RSA key of a GnuPG user.

While the above named vulnerabilities indicate that much work is left to improve the security of Intel SGX, it is nevertheless a promising technology for protection of code and data on remote platforms.

VIII. Conclusion

We presented a mechanism for verification of VNF integrity allowing to protect the VNF authentication credentials. We outlined the design principles and described the detailed design and implementation of the prototype. The evaluation showed that the prototype does not introduce any overhead beyond the initial bootstrap phase. By combining Intel SGX and operating system-level measurement (Linux IMA), we were able to build a more complex system with advanced security guarantees. The proposed prototype is under continuous development and we intend to add support for multiple containers and for storing integrity measurements in a hardware root of trust (e.g. TMP).

IX. Acknowledgements

This research has been performed within 5G-ENSURE project (www.5GEnsure.eu) and received funding from the European Unions Horizon 2020 research and innovation programme under grant agreement No 671562.

References

[1] SDN architecture. Technical Report ONF TR-50, Open Networking Foundation, June 2014.

[2] E. Haleplidis et al. Software-Defined Networking (SDN): Layers and Architecture Terminology. RFC 7426, RFC Editor, January 2015.

[3] D. Kreutz et al. Software-Defined Networking: A Comprehensive Survey. *Proceedings of the IEEE*, 103:14–76, Jan 2015.

[4] D. Kreutz et al. Towards Secure and Dependable Software-defined Networks. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 55–60. ACM, 2013.

[5] S. Scott-Hayward et al. A Survey of Security in Software Defined Networks. *IEEE Communications Surveys and Tutorials*, 18(1):623–654, 2016.

[6] M. Dhawan et al. SPHINX: Detecting Security Attacks in Software-Defined Networks. In *22nd Annual Network and Distributed System Security Symposium NDSS*. The Internet Society, 2015.

[7] C. Banse and S. Rangarajan. A Secure Northbound Interface for SDN Applications. In *IEEE Trustcom/BigDataSE/ISPA*, volume 1, pages 834–839, August 2015.

[8] R. Bonafiglia et al. Offloading personal security applications to a secure and trusted network node. In *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*, pages 1–2. IEEE, April 2015.

[9] T. Wood et al. Toward a software-based network: Integrating software defined networking and network function virtualization. *IEEE Network*, 29:36–41, May 2015.

[10] S. Soltesz et al. Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors. *SIGOPS Oper. Syst. Rev.*, 41:275–287, March 2007.

[11] Information technology-Trusted Platform Module-Part 1: Overview. Standard, International Organization for Standardization/International Electrotechnical Commission ISO/IEC 11889-1:2009(E), May 2009.

[12] Ittai Anati, Shay Gueron, Simon Johnson, and Vincent Scarlata. Innovative technology for CPU based attestation and sealing. In *Proc. 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, HASP '13, page 10. ACM, June 2013.

[13] D. Dolev and A. Yao. On the Security of Public Key Protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, Mar 1983.

[14] N. Paladi, M. Aslam, and C. Gehrmann. Trusted geolocation-aware data placement in infrastructure clouds. In *Proc. 13th International Conference on Trust, Security and Privacy in Computing and Communications*, TrustCom '14, pages 352–360. IEEE, September 2014.

[15] B. Parno. Bootstrapping Trust in a "Trusted" Platform. In *Proceedings of the 3rd Conference on Hot Topics in Security*. USENIX Association, July 2008.

[16] Container and Kernel-Based Virtual Machine (KVM) Virtualization for Network Function Virtualization (NFV). whitepaper, Intel Corp., Aug 2015.

[17] H. Krawczyk et al. HMAC: Keyed-Hashing for Message Authentication. RFC 2104, RFC Editor, February 1997.

[18] N. Paladi and C. Gehrmann. *TruSDN: Bootstrapping Trust in Cloud Network Infrastructure*, pages 104–124. SecureComm 2016. Springer International Publishing, 2017.

[19] M. Shih et al. S-NFV: Securing NFV States by Using SGX. In *Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, SDN-NFV Security '16, pages 45–48. ACM, 2016.

[20] N. Weichbrodt et al. AsyncShock: Exploiting Synchronisation Bugs in Intel SGX Enclaves. In *European Symposium on Research in Computer Security*, pages 440–457. Springer, 2016.

[21] R. Buhren et al. Fault Attacks on Encrypted General Purpose Compute Platforms. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, CODASPY '17, pages 197–204. ACM, 2017.